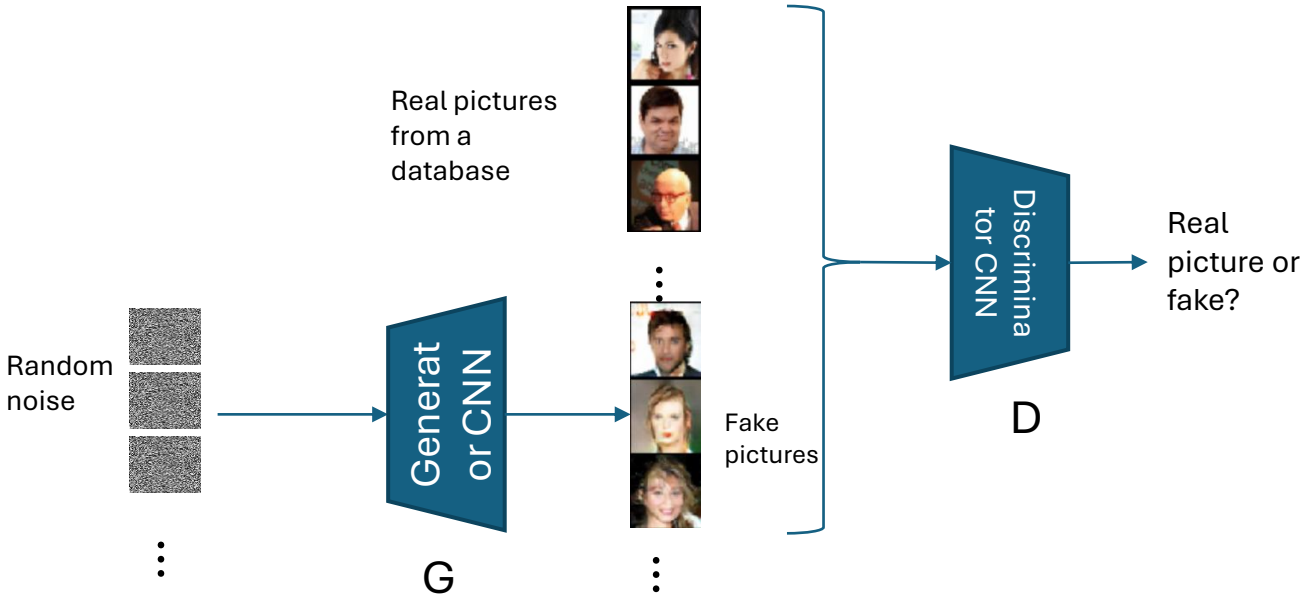
GANs and Diffusion Models and Some Math Primer

Nilanjan Ray

Excellent resources on generative models

- https://www.arxiv.org/abs/2510.21890
- https://cvpr2023-tutorial-diffusion-models.github.io/

Generative Adversarial Networks (GANs)…



Two-player game between the generator CNN and the discriminator CNN

We need to train both these CNNs so that they become good at their own trades!

$$min_G max_D E_{I\sim Real}[\log(D(I))] + E_{z\sim Noise}[\log(1 - D(G(z)))]$$

https://www.youtube.com/watch?v=QyQGevCAsr0

GANs training algorithm

for number of training iterations **do**

  for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

After training, use generator CNN to generate pictures!

Source: http://cs231n.stanford.edu/slides/2022/lecture_13_jiajun.pdf

GANs to WGANs

1️⃣ The Goal: Make Generated Data Match Real Data

We have

- $P_{\text{data}}(x)$: real image distribution
- $P_G(x)$: generated image distribution from $G(z),\ z \sim P_z$

We want

$$P_G \approx P_{\text{data}}$$

by minimizing some notion of **distance** between them.

2️⃣ The Original GAN Idea

Try to minimize a divergence like **KL** or **JS**:

$$D_{\text{KL}}(P_{\text{data}} \| P_G) = \mathbb{E}_{x \sim P_{\text{data}}}\left[\log \frac{P_{\text{data}}(x)}{P_G(x)}\right]$$

...but neither $P_{\text{data}}(x)$ nor $P_G(x)$ are known in closed form!

So, the **GAN trick** is to learn a discriminator $D(x)$ that *implicitly* measures how separable they are:

$$\min_G \max_D \ \mathbb{E}_{x \sim P_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim P_z}[\log(1 - D(G(z)))]$$

GANs to WGANs...

3️⃣ Problem: Divergences Are Not Stable When Supports Don't Overlap

When the fake samples lie far from real data (no overlap),
$D(x)$ becomes perfect → generator gradients vanish → **no learning signal**.

This motivates finding a **distance measure** that still gives gradients even when the distributions are far apart.

4️⃣ The Earth Mover's (Wasserstein) Distance — Intuition

Think of P_{data} and P_G as piles of **earth** (mass).
The Wasserstein distance is the **minimum work** needed to move one pile to match the other.

$$W(P_{\text{data}}, P_G) = \inf_{\gamma \in \Pi(P_{\text{data}}, P_G)} \mathbb{E}_{(x,y)\sim\gamma}\left[\, \|x - y\| \,\right]$$

where $\gamma(x, y)$ defines *how much mass* we transport from x to y.

GANs to WGANs…

5️⃣ Computing It in Practice

- **1D case (easy):**

  Sort both samples $x_{(i)} \sim P_{\text{data}}$ and $y_{(i)} \sim P_G$.

  Then the 1-Wasserstein distance is simply:

$$W_1(P_{\text{data}}, P_G) = \frac{1}{n} \sum_i |x_{(i)} - y_{(i)}|$$

  (just average distance after sorting!)

- **Higher-D case (hard):**

  We need to find the **minimum-cost matching** between real and fake samples — a *bipartite assignment problem* solved by the **Hungarian algorithm**, which is $O(n^3)$ and impractical for large datasets.

- **Moreover:**

  In high-dimensional spaces (like images), Euclidean distances become **unreliable**: all points are almost equally far apart ("curse of dimensionality").

  → So, we can't rely on direct pairwise matching.

GANs to WGANs...

6️⃣ Kantorovich–Rubinstein Duality to the Rescue

Instead of explicitly matching points, the duality theorem says we can compute it via a **function optimization**:

$$W(P_{\text{data}}, P_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_{\text{data}}}[f(x)] - \mathbb{E}_{x \sim P_G}[f(x)]$$

- f: any 1-Lipschitz function (slope ≤ 1 everywhere)
- The difference in expectations gives the **Earth Mover distance**

This version is **tractable** — we can **learn** f using a neural network!

7️⃣ The WGAN Formulation

Replace the discriminator with a **critic** $f_\omega(x)$ that outputs *any real number*:

$$\min_G \max_{\|f_\omega\|_L \leq 1} \mathbb{E}_{x \sim P_{\text{data}}}[f_\omega(x)] - \mathbb{E}_{z \sim P_z}[f_\omega(G(z))]$$

- Critic ≈ estimator of Wasserstein distance
- Generator tries to minimize it
- 1-Lipschitz constraint enforced by:
  - **Weight clipping**, or
  - **Gradient penalty**

Wasserstein GANs algorithm

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

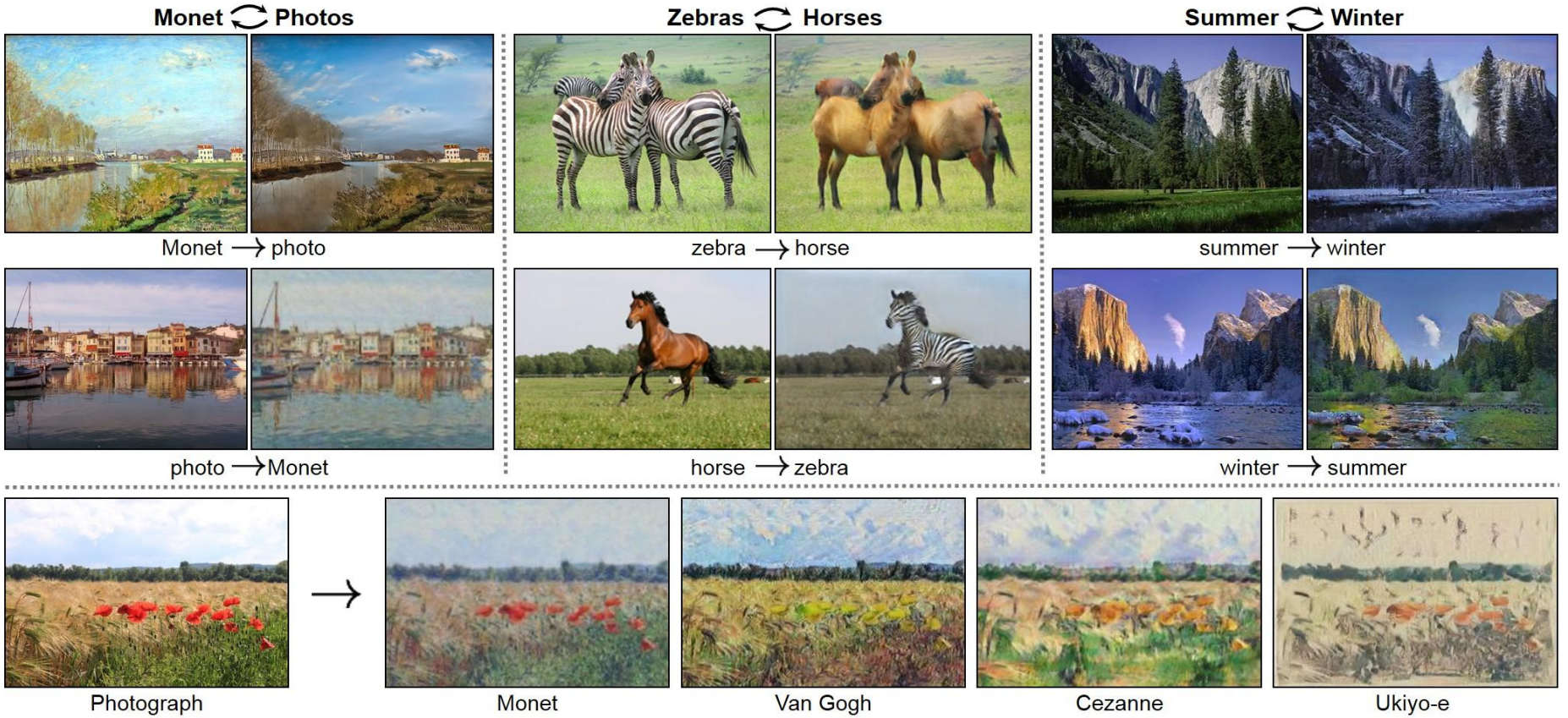
Typo/notation: g_w on the LHS should be denoted as del_w

g_theta on LHS should be denoted as del_theta

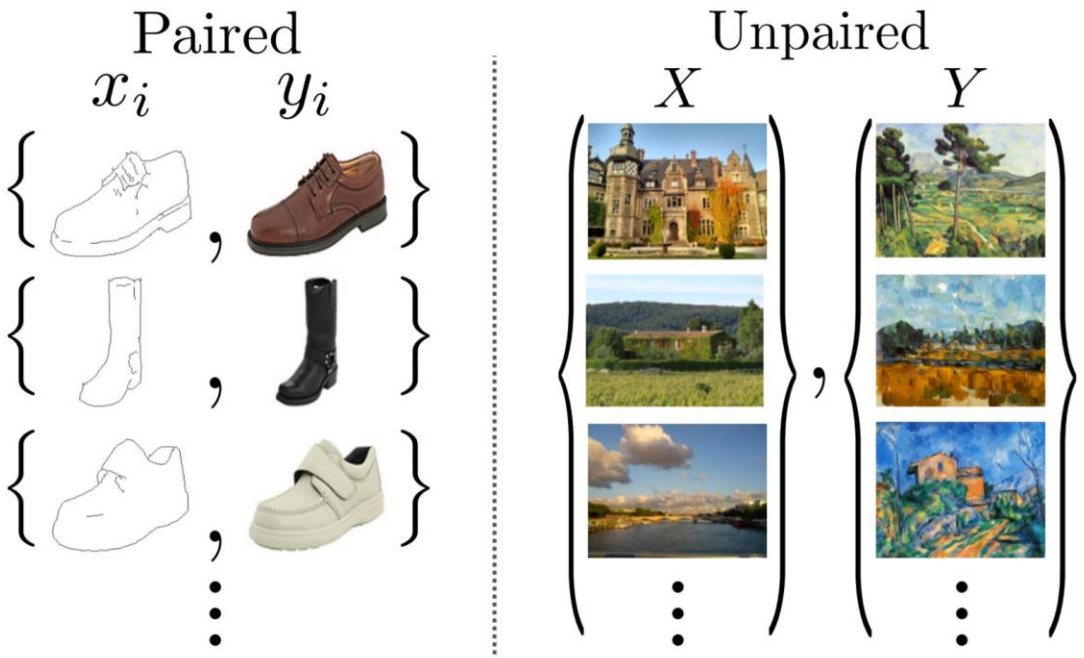
https://arxiv.org/pdf/1701.07875.pdf https://github.com/safwankdb/Wasserstein-GAN

CycleGAN: Unpaired Image-to-Image Translation using GANs

https://junyanz.github.io/CycleGAN/

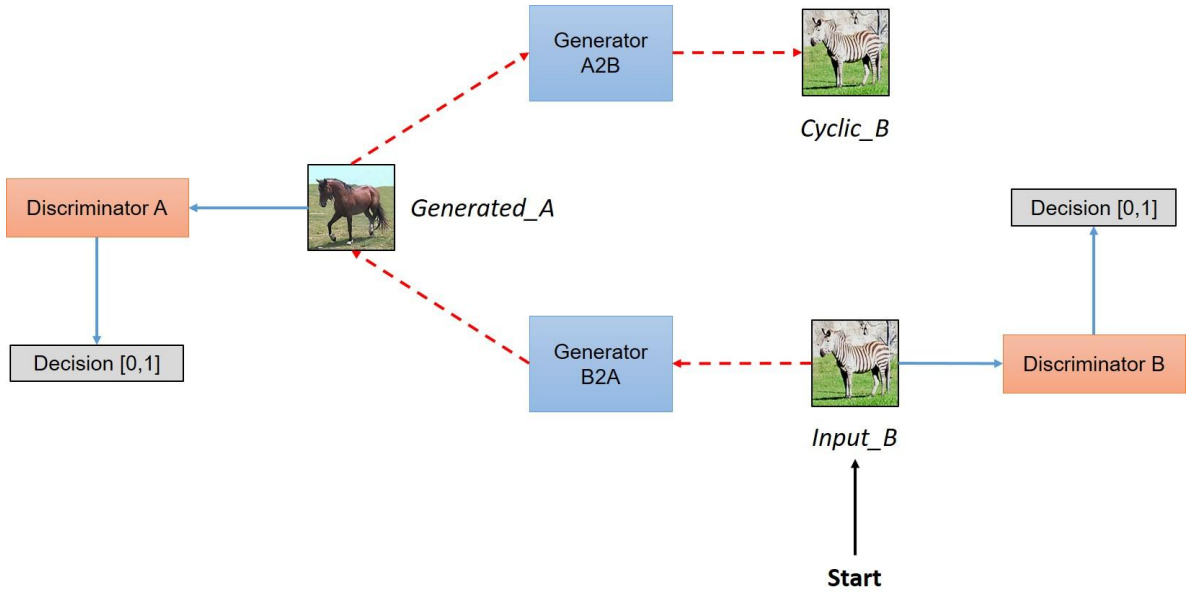


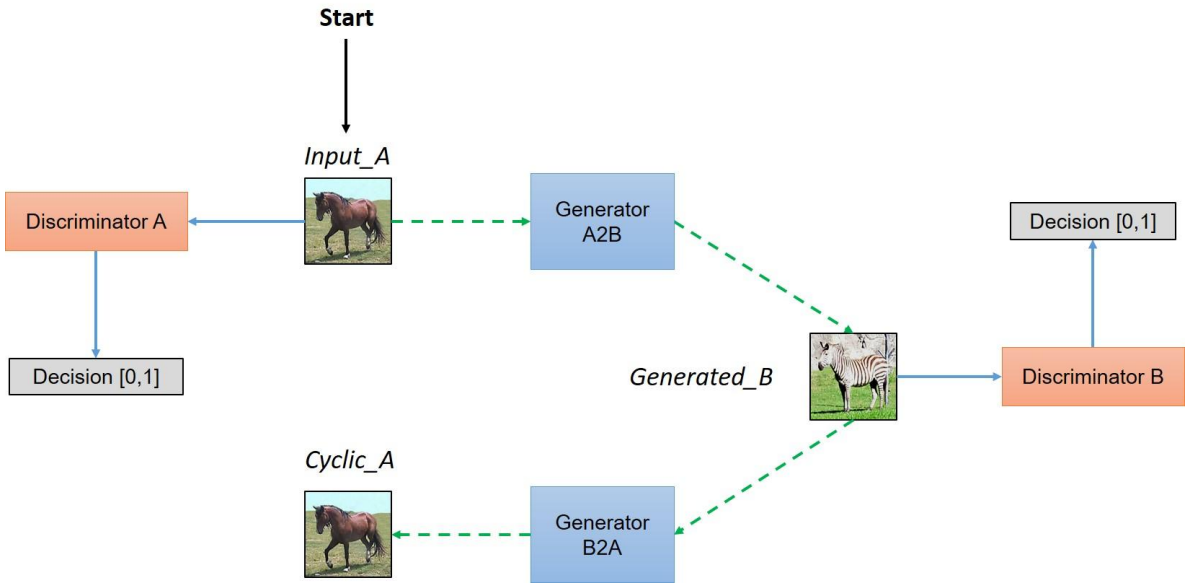
Unpaired image translation



CycleGAN schematic

https://hardikbansal.github.io/CycleGANBlog/





CycleGAN loss

Cycle consistency loss:

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_X(x)}[||F(G(x)) - x||_1] + \mathbb{E}_{y \sim p_Y(y)}[||G(F(y)) - y||_1]$$

Adversarial losses:

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_Y(y)}[logD_Y(y)] + \mathbb{E}_{x \sim p_X(x)}[log(1 - D_Y(G(x))]$$

$$\mathcal{L}_{GAN}(F, D_X, X, Y) = \mathbb{E}_{x \sim p_X(x)}[logD_X(x)] + \mathbb{E}_{y \sim p_Y(y)}[log(1 - D_X(F(y))]$$

Total loss:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F)$$

CycleGAN training algorithm

http://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/assignments/a4-handout.pdf

Algorithm 2 CycleGAN Training Loop Pseudocode

procedure TRAINCYCLEGAN

Draw a minibatch of samples $\{x^{(1)}, \ldots, x^{(m)}\}$ from domain X

Draw a minibatch of samples $\{y^{(1)}, \ldots, y^{(m)}\}$ from domain Y

Compute the discriminator loss on real images:

$$\mathcal{J}^{(D)}_{real} = \frac{1}{m}\sum_{i=1}^{m}(D_X(x^{(i)}) - 1)^2 + \frac{1}{n}\sum_{j=1}^{n}(D_Y(y^{(j)} - 1)^2$$

Compute the discriminator loss on fake images:

$$\mathcal{J}^{(D)}_{fake} = \frac{1}{m}\sum_{i=1}^{m}(D_Y(G_{X\to Y}(x^{(i)})))^2 + \frac{1}{n}\sum_{j=1}^{n}(D_X(G_{Y\to X}(y^{(j)})))^2$$

Update the discriminators

Compute the $Y \to X$ generator loss:

$$\mathcal{J}^{(G_{Y\to X})} = \frac{1}{n}\sum_{j=1}^{n}(D_X(G_{Y\to X}(y^{(j)})) - 1)^2 + \mathcal{J}^{(Y\to X\to Y)}_{cycle}$$

Compute the $X \to Y$ generator loss:

$$\mathcal{J}^{(G_{X\to Y})} = \frac{1}{m}\sum_{i=1}^{m}(D_Y(G_{X\to Y}(x^{(i)})) - 1)^2 + \mathcal{J}^{(X\to Y\to X)}_{cycle}$$

Update the generators

Problems in the GANs family

1 Training Instability

- GANs solve a **min–max game**:

$$\min_G \max_D V(D, G)$$

  The generator and discriminator continuously chase each other's gradients.
- Small learning-rate or architecture differences can destabilize training.
- Loss curves **do not correlate** reliably with sample quality.

2 Mode Collapse

- Generator learns to produce only a **subset of modes** in P_{data}:
  - E.g., only digits "1" and "7" instead of all MNIST digits.
- Happens when G finds a few "safe" outputs that always fool D.
- Symptoms:
  - High visual quality, **low diversity**
  - Repeated or identical samples
- Mitigations:
  - Feature matching
  - Mini-batch discrimination
  - Unrolled GAN, or WGAN-GP for smoother gradients

3 Vanishing or Exploding Gradients

- When D becomes too strong, ∇_G from $\log(1 - D(G(z))) \approx 0$
  → **no learning signal** for the generator.
- Conversely, poor normalization or learning rate can blow up gradients.
- Solutions:
  - Use **Wasserstein loss**, **spectral normalization**, **gradient penalty**, or **balanced updates (n_critic steps)**.

4 Non-Convergence

- Even if losses appear stable, P_G may oscillate around P_{data} without converging.
- Adversarial training ≠ convex optimization; there's no guarantee of global optimum.

Problems in the GANs family...

5 Evaluation Difficulty

- No direct log-likelihood or test set accuracy.
- Metrics like **FID** or **IS** are only **proxies** for realism and diversity.
- Human perceptual judgment is often still needed.

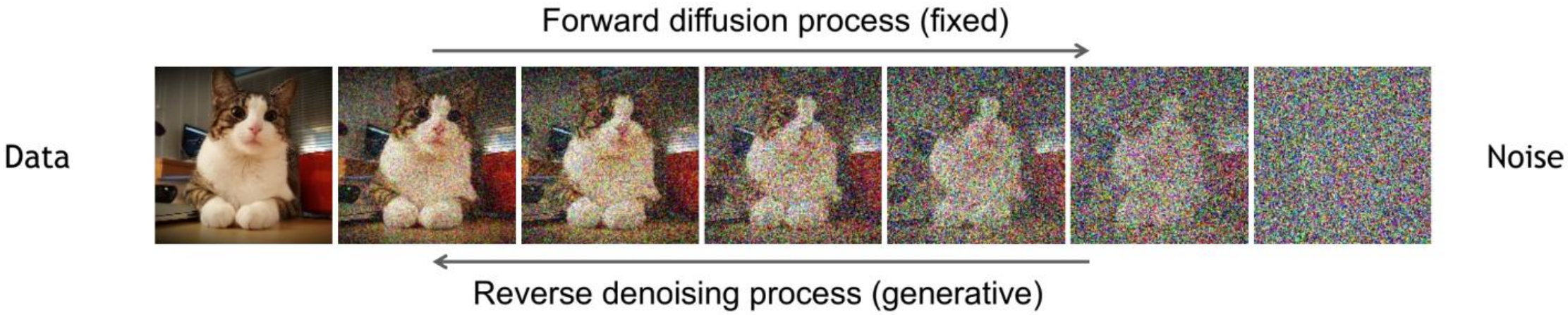
---

6 Resource Intensity

- High-resolution GANs (e.g., StyleGAN) require:
  - Large GPU memory
  - Millions of iterations
  - Careful architecture tuning (progressive growing, equalized learning rate, etc.)

Diffusion models

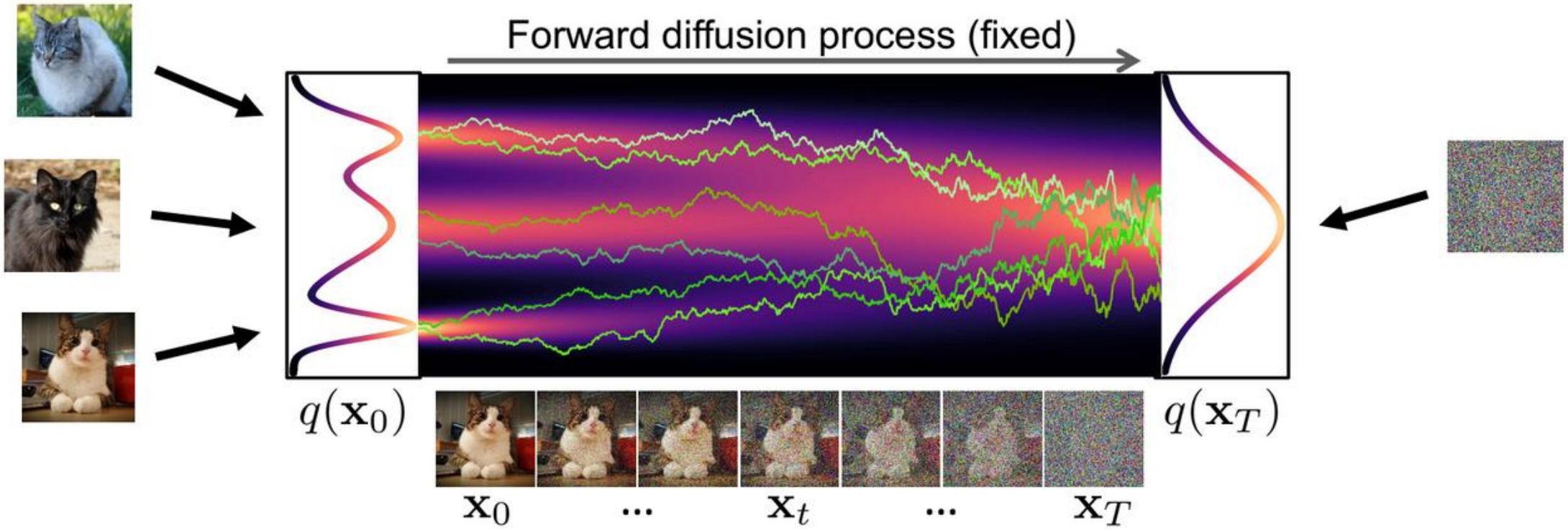
- Also known as denoising diffusion models (**DDPM**): https://proceedings.neurips.cc/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf
- There are several variants. The state-of-the-art models for image generation are based on DDPM.
- **DALL·E 2**, **Imagen (Google)**, **Stable Diffusion**, and **Midjourney** all use diffusion models.



How does DDPM work?

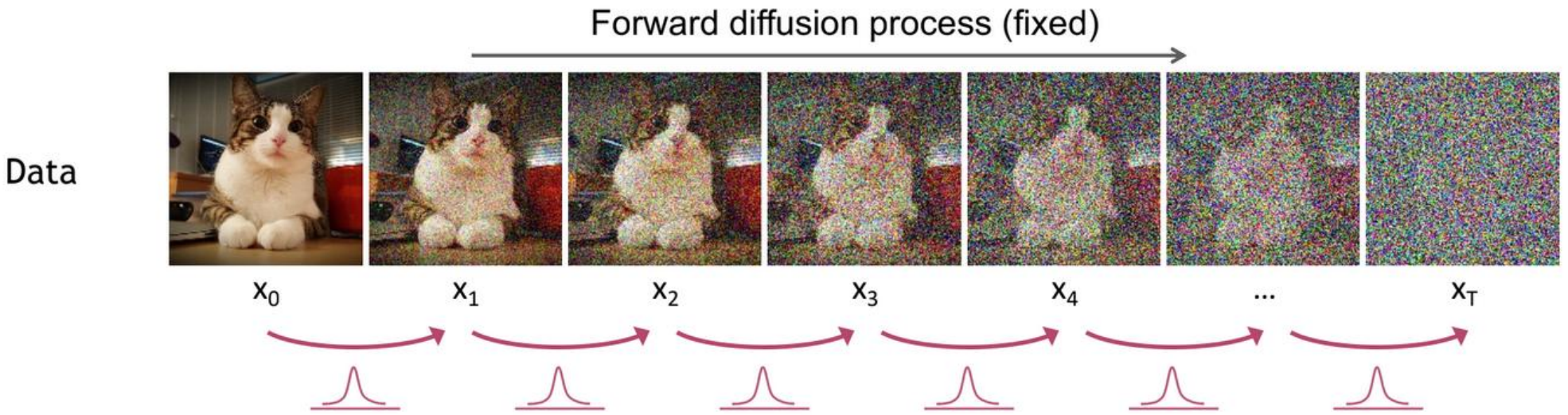
- Forward diffusion process gradually adds noise to an image until it becomes indistinguishable from pure noise
- Backward (generative) diffusion process learns to remove the noise gradually

Forward process: Data distribution to Gaussian distribution



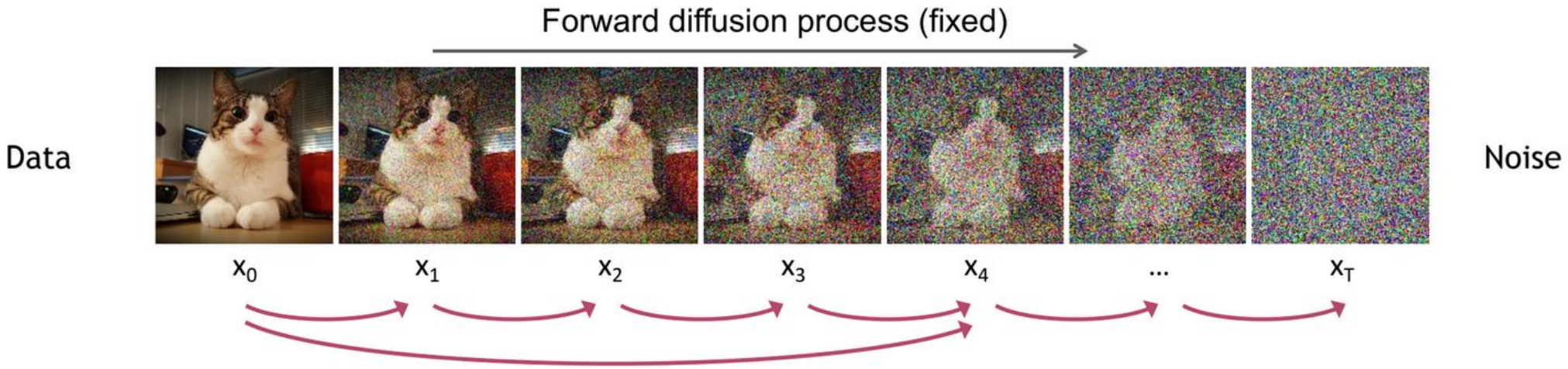
Forward diffusion

The formal definition of the forward process in T steps:



$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \quad \Rightarrow \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \qquad \text{(joint)}$$

Forward diffusion: Quite efficient



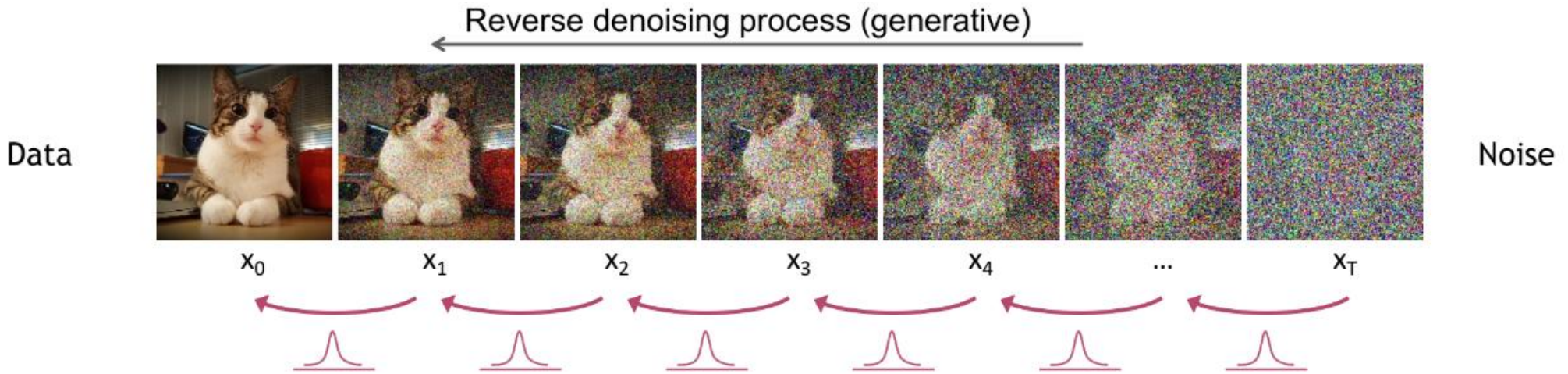
Define $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ ➡ $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$ (Diffusion Kernel)

For sampling: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\,\epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

β_t values schedule (i.e., the noise schedule) is designed such that $\bar{\alpha}_T \to 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Reverse denoising process (generative)

Formal definition of forward and reverse processes in T steps:



$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t)}_{\text{Trainable network}}, \sigma_t^2\mathbf{I})$$

$$\Rightarrow \quad p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

Training and inference algorithms

Algorithm 1 Training

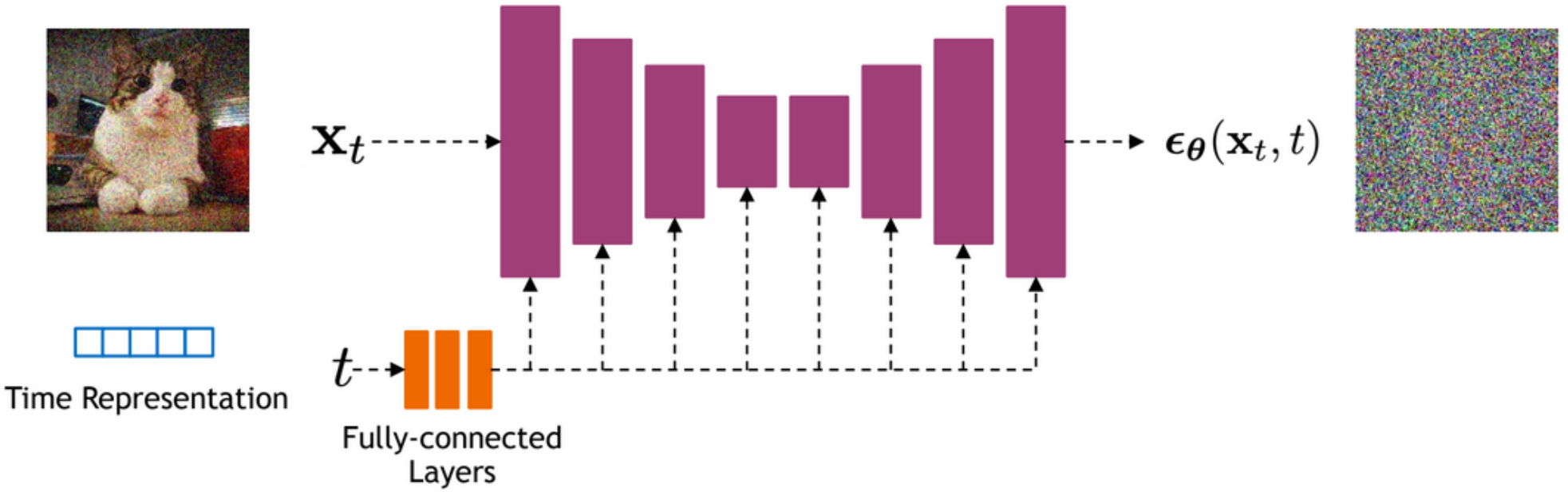
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Time-embedded architectures for DDPM

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent $\epsilon_\theta(\mathbf{x}_t, t)$



Evaluating generative models

1️⃣ Inception Score (IS)

Goal: Measure both **image quality** and **diversity**

- Uses a pretrained **Inception-V3** network
- Computes class probabilities $p(y|x)$ for generated images
- Good models: confident & varied predictions

$$\text{IS} = \exp\left(\mathbb{E}_x[D_{\text{KL}}(p(y|x) \parallel p(y))]\right)$$

Interpretation:

- Higher IS → images are clearer and more diverse
- Weakness: ignores real data (only tests generator output)

2️⃣ Fréchet Inception Distance (FID)

Goal: Compare generated & real feature distributions

- Extract **Inception-V3** features from both real and fake images
- Model features as Gaussians: (μ_r, Σ_r), (μ_g, Σ_g)
- Compute 2-Wasserstein distance between them:

$$\text{FID} = ||\mu_r - \mu_g||^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}\right)$$

Interpretation:

- Lower FID → generated distribution closer to real one
- Sensitive to image quality, mode collapse, and diversity

Math primers needed for some upcoming generative models

- Maximum likelihood estimation (MLE)
- Ordinary differential equations (ODE)
- Transformation of random variables

MLE

🔢 1️⃣ Maximum Likelihood Estimation (MLE)

We start with **a parametric model** $p(x;\theta)$
and a dataset $\{x_i\}_{i=1}^N$ assumed **i.i.d.** from that model.

The **likelihood** is:

$$L(\theta) = \prod_{i=1}^{N} p(x_i;\theta)$$

and we usually work with the **log-likelihood**:

$$\ell(\theta) = \sum_{i=1}^{N} \log p(x_i;\theta)$$

The MLE is:

$$\hat{\theta} = \arg\max_{\theta} \ell(\theta)$$

Example: 1D Gaussian

$$p(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

The log-likelihood:

$$\ell(\mu,\sigma) = -N\log\sigma - \frac{1}{2\sigma^2}\sum_i (x_i-\mu)^2 + \text{const.}$$

Set derivatives to zero:

$$\frac{\partial \ell}{\partial \mu} = 0 \Rightarrow \hat{\mu} = \frac{1}{N}\sum_i x_i$$

$$\frac{\partial \ell}{\partial \sigma} = 0 \Rightarrow \hat{\sigma}^2 = \frac{1}{N}\sum_i (x_i-\hat{\mu})^2$$

✅ This is how MLE "learns" parameters by maximizing data likelihood — the same principle flows and VAEs use, but in much higher dimensions.

ODEs

🧩 2️⃣ Ordinary Differential Equations (ODEs)

A simple 1D ODE defines how a variable $y(t)$ evolves with time:

$$\frac{dy}{dt} = f(y, t)$$

Given an initial condition $y(0) = y_0$, the **solution** is the curve that satisfies this equation.

Example

$$\frac{dy}{dt} = -2y, \quad y(0) = 3$$

Analytic solution:

$$y(t) = 3e^{-2t}$$

At $t = 0.5$, $y = 3e^{-1} \approx 1.10$

Euler's Method (numerical)

We can approximate the solution iteratively:

$$y_{t+\Delta t} = y_t + \Delta t\, f(y_t, t)$$

Example:

$$y_{t+1} = y_t - 2\Delta t\, y_t$$

If $\Delta t = 0.1$:

$$y_1 = 3 - 0.6 = 2.4,\ y_2 = 2.4 - 0.48 = 1.92,\ \text{etc.}$$

✅ Euler's method is the foundation of how **Neural ODEs** integrate continuous dynamics of hidden states.

Transformation of R.V.s

🔁 3️⃣ Random Variable Transformation (1D Case)

Suppose $Z \sim p_Z(z)$, and we define
$X = g(Z)$, where g is **invertible** and **differentiable**.

Then the probability densities satisfy:

$$p_X(x) = p_Z(g^{-1}(x)) \left| \frac{dg^{-1}(x)}{dx} \right|$$

Example

Let $Z \sim \mathcal{N}(0, 1)$, and define $X = g(Z) = 2Z + 3$.

Then:

$$g^{-1}(x) = \frac{x-3}{2}, \quad \frac{dg^{-1}(x)}{dx} = \frac{1}{2}$$

$$p_X(x) = \frac{1}{2} p_Z\left(\frac{x-3}{2}\right)$$

✅ So the mean becomes 3, the std becomes 2:
the Gaussian transforms exactly as expected.

🌐 4️⃣ Random Variable Transformation (n-D Case)

For a multivariate invertible transform $\mathbf{x} = g(\mathbf{z})$:

$$p_X(\mathbf{x}) = p_Z(g^{-1}(\mathbf{x})) \left| \det \frac{\partial g^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right|$$

Equivalently (by inverting the Jacobian):

$$\log p_X(\mathbf{x}) = \log p_Z(\mathbf{z}) - \log \left| \det \frac{\partial g(\mathbf{z})}{\partial \mathbf{z}} \right|$$

This is the **change-of-variables formula** — the cornerstone of **normalizing flows**.

✅ Interpretation

- $p_Z(\mathbf{z})$: simple base distribution (e.g., Gaussian)
- $g(\mathbf{z})$: invertible neural network transforming latent to data space
- The Jacobian determinant adjusts the density for stretching/ compression of space.

→ If we can compute $\det J$, we can do **exact MLE** on high-dimensional data.

Example: 2D Transformation and the Jacobian Determinant

Suppose we start with a 2D random variable

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \sim p_Z(\mathbf{z})$$

and define an **invertible transformation** $\mathbf{x} = g(\mathbf{z})$:

$$\begin{cases} x_1 = 2z_1 + z_2 \\ x_2 = z_1 + 3z_2 \end{cases}$$

This is a **linear transformation**, so we can write:

$$\mathbf{x} = A\mathbf{z}, \quad A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

2D Example...

1 Compute the Jacobian

The **Jacobian matrix** $J_g(\mathbf{z}) = \frac{\partial \mathbf{x}}{\partial \mathbf{z}}$ is:

$$J_g(\mathbf{z}) = \begin{bmatrix} \frac{\partial x_1}{\partial z_1} & \frac{\partial x_1}{\partial z_2} \\ \frac{\partial x_2}{\partial z_1} & \frac{\partial x_2}{\partial z_2} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

(same as A since this transform is linear)

2 Determinant of the Jacobian

$$\det(J_g) = (2)(3) - (1)(1) = 6 - 1 = 5$$

This tells us that:

- The transformation **expands area** by a factor of 5
- i.e., one unit square in z-space maps to an area 5× larger in x-space

3 Log Determinant

$$\log |\det(J_g)| = \log 5 \approx 1.609$$

The log-determinant term will appear in the log-likelihood when transforming densities.

4 Inverse Transformation

We can compute g^{-1} since A is invertible:

$$A^{-1} = \frac{1}{5} \begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix}$$

So:

$$\begin{cases} z_1 = \frac{1}{5}(3x_1 - x_2) \\ z_2 = \frac{1}{5}(-x_1 + 2x_2) \end{cases}$$

2D Example...

5️⃣ Density Transformation

If $\mathbf{z} \sim p_Z(\mathbf{z})$, then the new density $p_X(\mathbf{x})$ is:

$$p_X(\mathbf{x}) = p_Z(g^{-1}(\mathbf{x})) \, |\det J_{g^{-1}}(\mathbf{x})|$$

But since $J_{g^{-1}} = (J_g)^{-1}$,

$$|\det J_{g^{-1}}| = \frac{1}{|\det J_g|} = \frac{1}{5}$$

So:

$$p_X(\mathbf{x}) = \frac{1}{5} p_Z(g^{-1}(\mathbf{x}))$$

and

$$\log p_X(\mathbf{x}) = \log p_Z(g^{-1}(\mathbf{x})) - \log 5$$

✅ The second term, $-\log |\det J_g|$, is the **volume correction** term that normalizing flows use in log-likelihood computation.

6️⃣ Geometric Intuition

- Determinant $= 5 \rightarrow$ the transformation *stretches* 2D space 5×
- If p_Z is a Gaussian centered at (0,0), the transformed p_X is an **elliptical Gaussian** stretched along the eigenvectors of A.

Optional (Nonlinear Extension)

If we had:

$$x_1 = 2z_1 + \sin(z_2), \quad x_2 = z_2^2$$

then the Jacobian would depend on z:

$$J_g = \begin{bmatrix} 2 & \cos(z_2) \\ 0 & 2z_2 \end{bmatrix}$$

$$\det(J_g) = 4z_2 \quad \text{and} \quad \log|\det(J_g)| = \log|4z_2|$$

which illustrates how **flows handle local stretching/compression** at each point.